Задание 1. Теоретическая часть

1. Определения

Что такое измерительные методы оценки программ?

Измерительные методы оценки программ — это совокупность техник и инструментов, позволяющих количественно определить характеристики программного обеспечения, такие как размер, сложность, качество и эффективность. Они основаны на сборе и анализе метрик, что помогает объективно оценить состояние и качество программных продуктов.

Какие основные цели их применения?

- \*\*Контроль качества\*\*: выявление слабых мест и потенциальных ошибок.

- \*\*Планирование и управление проектами\*\*: оценка объема работы, сложности и затрат.

- \*\*Оптимизация разработки\*\*: выявление сложных участков кода для улучшения.

- \*\*Обеспечение качества сопровождения\*\*: мониторинг изменений и их влияния на проект.

2. Примеры метрик и их назначение

|  |  |  |
| --- | --- | --- |
| Тип метрики | Примеры | Назначение |
| Метрики размера | Количество строк кода, число функций | Оценка объема работы, сравнение различных версий, контроль роста проекта |
| Метрики сложности | Цикломатическая сложность | Оценка сложности тестирования и поддержки, выявление потенциальных ошибок и уязвимых участков |
| Метрики качества | Коэффициент сопровождения, покрытие тестами | Оценка надежности, тестируемости, уровня поддержки и качества программного продукта |

3. Условия применения

В каких случаях применяются метрики?

- В процессе разработки для оценки текущего состояния проекта.

- При проведении ревью и аудита кода.

- В автоматизированных тестах и CI/CD-процессах.

- Для оценки эффективности работы команды и качества программного обеспечения.

- В процессе сопровождения и модернизации системы.

Какие ограничения могут быть у измерительных методов?

- Метрики могут не учитывать контекст и специфику конкретного проекта.

- Высокая сложность метрик может затруднить их интерпретацию.

- Некоторые метрики могут давать искаженную картину при неправильном использовании.

- Не все аспекты качества можно количественно измерить, например, удобство использования или безопасность.

- Возможна зависимость результатов от выбранных методов измерения и их настроек.

Задание 2. Практическая часть

***Анализ кода 1***

python

# Импорт необходимых библиотек

import requests # Для выполнения HTTP-запросов к API ЦБ РФ

import pandas as pd # Для работы с данными в табличном формате

import matplotlib.pyplot as plt # Для построения графиков

from datetime import datetime, timedelta # Для работы с датами

import xml.etree.ElementTree as ET # Для парсинга XML-данных

# Список валют, которые мы будем анализировать (обновлен по заданию)

currencies = ['DKK', 'CAD', 'NZD', 'PLN'] # Датская крона, Канадский доллар, Новозеландский доллар, Злотый

# Создаем словарь для хранения данных по валютам

currency\_data = {currency: [] for currency in currencies}

# Устанавливаем диапазон дат: последние 30 дней от текущей даты

end\_date = datetime.now() # Текущая дата

start\_date = end\_date - timedelta(days=30) # Дата 30 дней назад

current\_date = start\_date # Начинаем с начальной даты

# Цикл по всем дням в диапазоне

while current\_date <= end\_date:

# Форматируем дату в формат, который понимает API ЦБ (дд/мм/гггг)

formatted\_date = current\_date.strftime('%d/%m/%Y')

# Формируем URL для запроса курсов валют на конкретную дату

url = f"https://www.cbr.ru/scripts/XML\_daily.asp?date\_req={formatted\_date}"

# Отправляем GET-запрос к API ЦБ

response = requests.get(url)

response.encoding = 'windows-1251' # Указываем кодировку ответа

# Проверяем, что запрос успешен (код 200)

if response.status\_code == 200:

# Парсим XML-ответ

root = ET.fromstring(response.content)

# Получаем дату из атрибутов XML-документа

date\_record = root.attrib.get('Date')

# Ищем все элементы 'Valute' в XML

for valute in root.findall('Valute'):

# Получаем буквенный код валюты

char\_code = valute.find('CharCode').text

# Проверяем, есть ли эта валюта в нашем списке

if char\_code in currencies:

# Получаем значение курса и номинал

value = float(valute.find('Value').text.replace(',', '.')) # Заменяем запятую на точку для float

nominal = int(valute.find('Nominal').text) # Номинал (например, 1, 10, 100 и т.д.)

# Нормализуем курс (приводим к стоимости за 1 единицу валюты)

normalized\_value = value / nominal

# Добавляем данные в словарь

currency\_data[char\_code].append({

'date': date\_record, # Дата записи

'value': normalized\_value # Нормализованное значение курса

})

# Переходим к следующему дню

current\_date += timedelta(days=1)

# Сохраняем данные в Excel-файл

with pd.ExcelWriter('currency\_rates.xlsx') as writer: # Создаем Excel-файл

for currency in currencies:

# Создаем DataFrame из собранных данных

df = pd.DataFrame(currency\_data[currency])

if not df.empty: # Проверяем, что данные есть

# Преобразуем дату в формат datetime и сортируем по дате

df['date'] = pd.to\_datetime(df['date'], format='%d.%m.%Y')

df.sort\_values('date', inplace=True)

# Сохраняем в отдельный лист Excel с именем валюты

df.to\_excel(writer, sheet\_name=currency, index=False)

# Строим графики для каждой валюты

for currency in currencies:

# Создаем DataFrame из собранных данных

df = pd.DataFrame(currency\_data[currency])

# Проверяем, есть ли данные

if df.empty:

print(f"Нет данных для {currency}.")

continue

# Преобразуем и сортируем даты

df['date'] = pd.to\_datetime(df['date'], format='%d.%m.%Y')

df.sort\_values('date', inplace=True)

# Создаем график

plt.figure(figsize=(12, 6))

plt.plot(df['date'], df['value'], marker='o', linestyle='-', linewidth=2, markersize=6, color='#2c5aa0')

plt.title(f'Динамика курса {currency} к рублю\n({start\_date.strftime("%d.%m.%Y")} - {end\_date.strftime("%d.%m.%Y")})', fontsize=14, pad=20)

plt.xlabel('Дата', fontsize=12)

plt.ylabel('Курс, руб', fontsize=12)

plt.xticks(rotation=45, fontsize=10)

plt.yticks(fontsize=10)

plt.grid(True, alpha=0.3)

plt.gca().xaxis.set\_major\_formatter(plt.matplotlib.dates.DateFormatter('%d.%m'))

plt.tight\_layout()

plt.savefig(f'{currency}\_trend.png', dpi=150)

plt.show()

print("Обработка завершена. Проверьте файлы:")

print("- currency\_rates.xlsx")

print("- Графики в формате PNG")

1. Метрики размера

|  |  |  |
| --- | --- | --- |
| Файл | Количество строк (без пустых и комментариев) | Кол-во функций |
| Код 1 | 108 | 0 (функций нет) |

2. Оценка сложности

*- Функции:* отсутствуют, значит, сложность для функций не определяем.

*- Общий код:* сложность можно оценить по количеству ветвлений и циклов. В основном, есть цикл `while`, внутри которого есть условие `if`. В целом, сложность невысокая, так как структура простая.

3. Выводы

**Самая сложная часть:** цикл `while` с вложенными условиями и парсинг XML. Внутри цикл по валютам, условия проверки и парсинг данных.

*Где могут возникнуть ошибки:*

- В случае отсутствия данных по валютам или неправильного формата XML.

- В случае ошибок сети при запросах.

- В случае некорректных данных (например, неправильный формат чисел).

***Анализ кода 2***

```python

def calculate\_sum(a, b):

return a + b

def factorial(n):

if n == 0:

return 1

else:

return n \* factorial(n - 1)

def print\_numbers(n):

for i in range(n):

if i % 2 == 0:

print(i, "is even")

else:

print(i, "is odd")

1. Метрики размера

|  |  |  |
| --- | --- | --- |
| Функция | Количество строк (без пустых и комментариев) | Кол-во функций |
| Calculate\_sum | 2 | 3 |
| Factorial | 8 | 3 |
| Print\_numbers | 8 | 3 |

2. Оценка сложности

|  |  |  |
| --- | --- | --- |
| Функция | Ветвления | Цикломатическая сложность |
| Calculate\_sum | 0 | 1 |
| Factorial | 1(условие if) | 2 |
| Print\_numbers | 1(условие if) внутри цикла | 2 |

3. Выводы

Самая сложная функция: `factorial` и `print\_numbers` обе имеют сложность 2, так как содержат по одному условию.

*Где могут возникнуть ошибки:*

- В `factorial`: при передаче отрицательных чисел — рекурсия может продолжаться бесконечно или вызвать ошибку.

- В `print\_numbers`: при передаче отрицательных значений `n` цикл не выполнится, ошибок не возникнет, но логика может быть неочевидной.